EE 3523 Final Project

Bruno E. Gracia Villalobos
Computer Engineering
Professor Grigoryan - UTSA

May 3, 2019

INTRODUCTION

One of my songs was used for the signal to be filtered.

Analyzed a sample of 40 seconds from 00:28:00 to 01:08:00

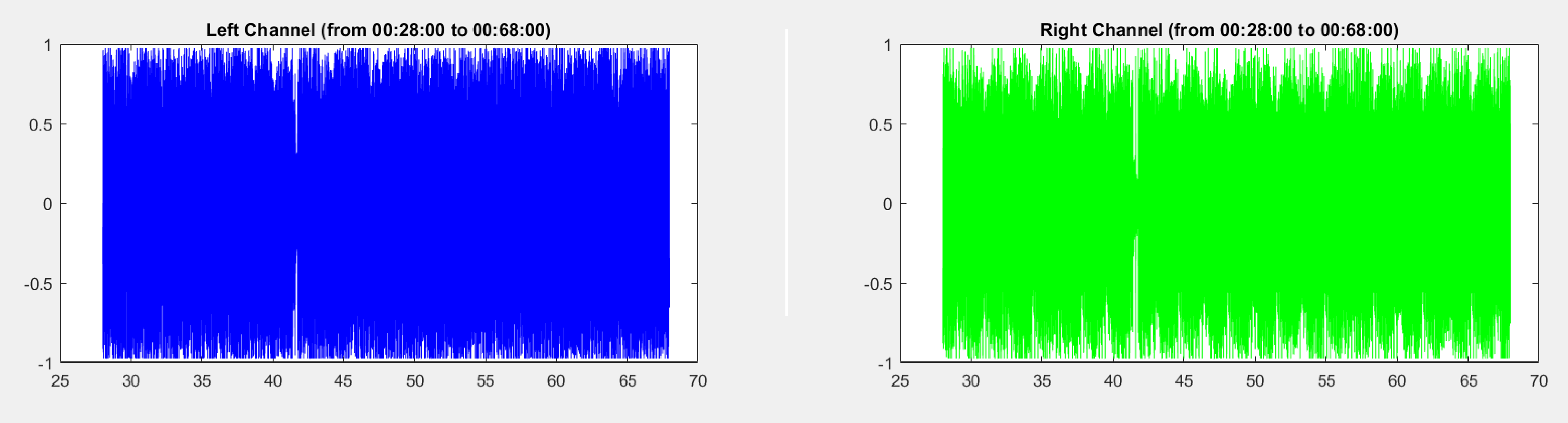
2 Channel, Fs = 48kHz, 1920000 samples per channel.

Both L and R channels are filtered for each step of the project.

```
%Read in my song "ascend - bruno G*"
% "native" => samples are 24 bits stored as int32's
[samples, fs] = audioread("ascend_1604.wav", "double");
info = audioinfo("ascend_1604.wav");
```

```
%Get L and R Channels from WAV and make into row
%get 40 seconds of my song
sampleRate = info.SampleRate;
begin = 28; %start at second 28
duration = 40;

lchannel = samples( (sampleRate*begin) : ...
    (sampleRate*(begin+duration))-1 ,1) .' ;

rchannel = samples( (48000*begin) : ...
    (sampleRate*(begin+duration))-1 ,2) .' ;

numSamples = length(lchannel);

%Assemble both channels into a track for playback
songSnippet = [ lchannel.' rchannel.'];

%play the audio tracks
%sound(songSnippet,sampleRate);
```

1. Code



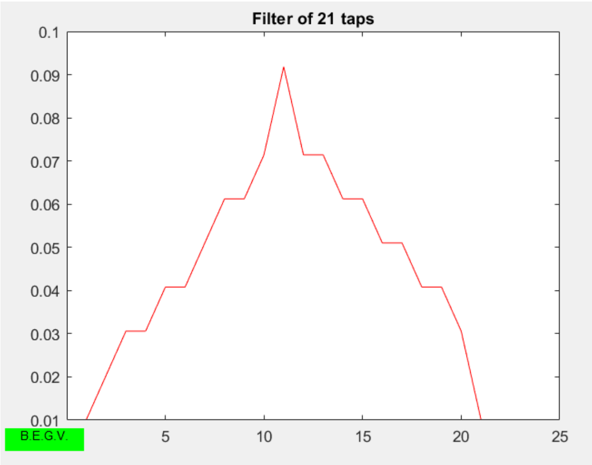
1. MATLAB Figures – raw data

1. Play the Song

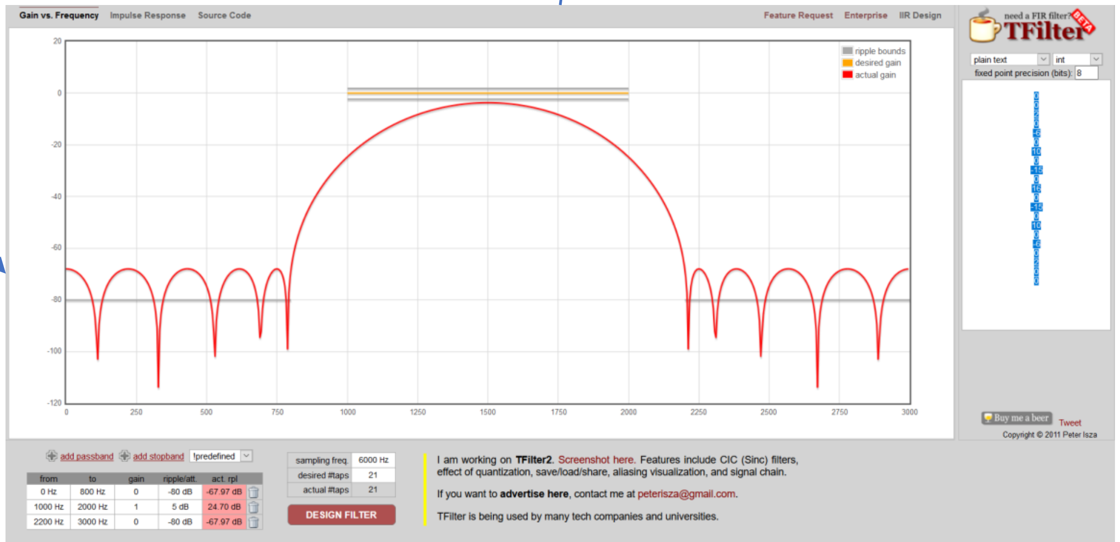
```
%Assemble both channels into a track for playback
songSnippet = [ lchannel.' rchannel.'];

%play the audio tracks
sound(songSnippet,sampleRate);
```

2. Filter the Signal

- Filter of 21 taps created given guidelines
- Normalized with k = sum of h(n)
- *other filters tested using online tap-filter generator
    - These are commented out



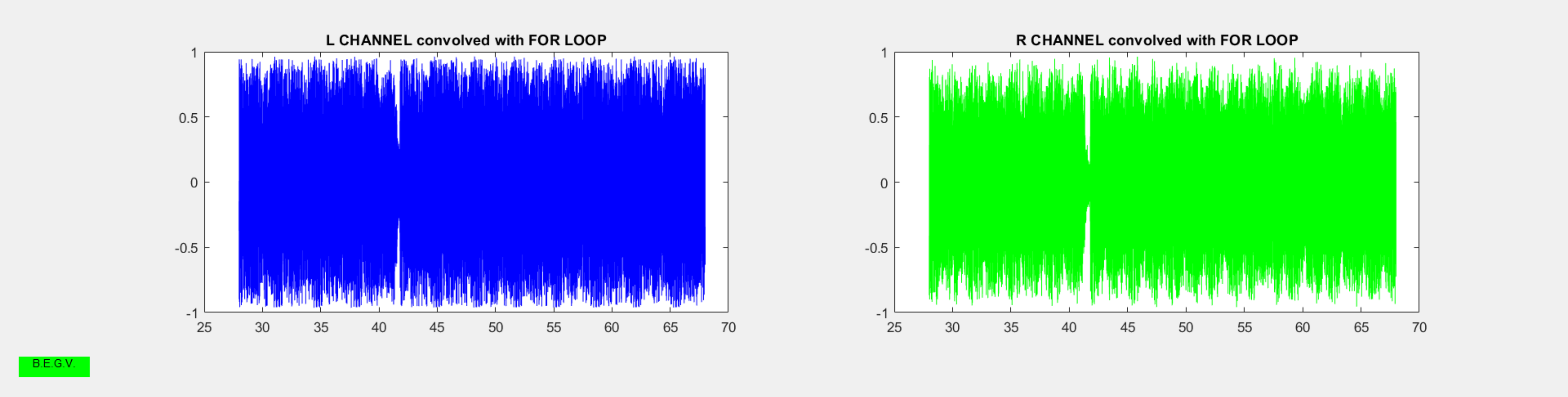

```
%Create Triangle Filter filter(11) is the center
filter = [1 2 3 3 4 4 5 6 6 7 9 7 7 6 6 5 5 4 4 3 1];
%{
filter = [-659 -1915 -2005 -358 1679 1089 -1853 -2807 ...
   2077 10186 14235 10186 2077 -2807 -1853 1089 ...
1679 -358 -2005 -1915 -659];
%}
%filter = [3 -1 -3 -1 -7 2 25 -3 -44 1 53 1 -44 -3 25 2 -7 -1 -3 -1 3];
%BPF 1-2KHZ
%filter = [0 0 2 0 -6 0 10 0 -15 0 16 0 -15 0 10 0 -6 0 2 0 0];

k = sum(filter); % normalize
filter = filter/k;
filterAxis = 1:1:length(filter);
```



2A. Calculate Direct Convolution

- First h(n) was mirrored
- Then for loop was generated to calculate the convolution
- Both L and R channels are convolved with the triangle filter

```
%LEFT channel
convLeft = lchannel;
convFilter = filter(length(filter) : -1 : 1); %mirror

for n=11:numSamples-10
    p=lchannel(n-10:n+10);
    convLeft(n)=sum(p.*convFilter);
end

%RIGHT channel
convRight = rchannel;

for n=11:numSamples-10
    p=rchannel(n-10:n+10);
    convRight(n)=sum(p.*convFilter);
end
```



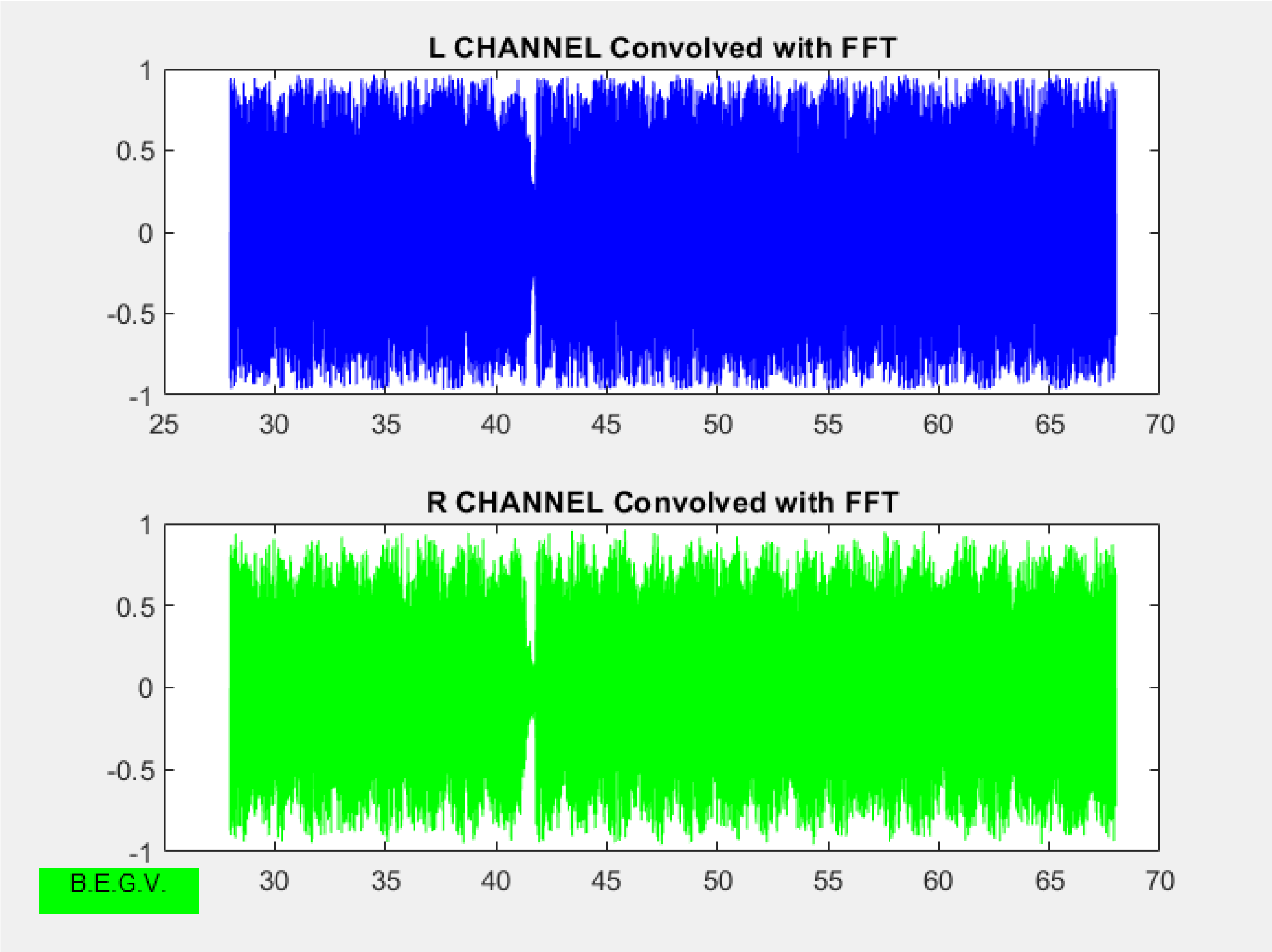
2A. MATLAB Figures

2A. Calculate Convolution with FFT

- This problem was confusing because part C asks to do FFT and IFFT for the convolution also.
- Steps:
    - Zero padding of L and R Channels with length of filter -1.
    - Zero padding filter with length of L channel or R channel.
    - Finding FFT's of L channel, R channel, and the triangle filter.
    - L channel * filter = F[L channel] x F[filter]
    - R channel * filter = F[R channel] x F[filter]
    - Inverse FFT of F[L channel] x F[filter]
    - Inverse FFT of F[R channel] x F[filter]

```
%%%%%%%%%%%%%%%Convolution by FFT%%%%%%%%%%%%

%Setup FFT of L and R Channels, and Filter
fft_l = fft( [ lchannel zeros(1, length(filter)-1) ] );
fft_r = fft( [ rchannel zeros(1, length(filter)-1) ] );
%fft_f = fft([filter zeros(1, length(lchannel) - length(filter)) ]); %not
%needed, automatic padding
fft_f = fft(filter, length(fft_l));

%Convolve in frequency domain
convF_l = fft_l .* fft_f;
convF_r = fft_r .*fft_f;

%Go back to time domain
conv_l = ifft(convF_l);
conv_r = ifft(convF_r);

%resize timeAxis to account for the added samples from convolution
timeAxis =  begin:(1/sampleRate):(begin+duration + (19/sampleRate));
```

2A. MATLAB Figures



2B-1. Overlap Add Convolution

- Both L and R channels were split into chunks of 10 sec each.
- Each channel has a 4 x (480000+20) matrix to store the 4 chunks
- 10 sec with 48000 sampling rate is 480000 samples

```
%%%%%%%%%%%%PART B-1%%%%%%%%%%%%%%%%%%%%

tenSec = sampleRate*10; %constant
chunks = 0: (tenSec): numSamples; %mark the bounds of each chunk
chunks = chunks + 1;
totChunks = length(chunks) - 1; %constant for total chunks

%set aside memory for overlap add method.
%this is a 4 row by 480000 + 20 zeros vector for each convolution
chunkV_l = zeros(totChunks, tenSec + (length(filter)-1) );
chunkV_r = chunkV_l;

%make loop to make a matrix to hold 10 second chunks
for i=1:totChunks
   %get the 10 sec chunks from song for each channel
   L = lchannel(chunks(i):chunks(i+1)-1);
   R = rchannel(chunks(i):chunks(i+1)-1);

   %assign to the first ten seconds the fetched samples
   chunkV_l(i,1:tenSec) = L;
   chunkV_r(i,1:tenSec) = R;
end
```

```
%%%%%%%%%%setup variables for FFT%%%%%%%%%%%%%%

%same as chunkV_r setup but for storing FFTs
chunkV_r_fft = zeros(totChunks, length(chunkV_l(1,:)));
chunkV_l_fft = chunkV_r_fft;

%get the frequency bins and store in another matrix
for i=1:totChunks
    chunkV_l_fft(i,:) = fft(chunkV_l(i,:));
    chunkV_r_fft(i,:) = fft(chunkV_r(i,:));
end

%now resize the filter vector for each tensec chunk
fft_f = fft(filter,length(chunkV_l_fft(1,:)));
```

chunkV_l	4x480020 double
chunkV_l_fft	4x480020 complex double
chunkV_r	4x480020 double
chunkV_r_fft	4x480020 complex double
fft_conv_r	1x1920020 double
fft_f	1x480020 complex double

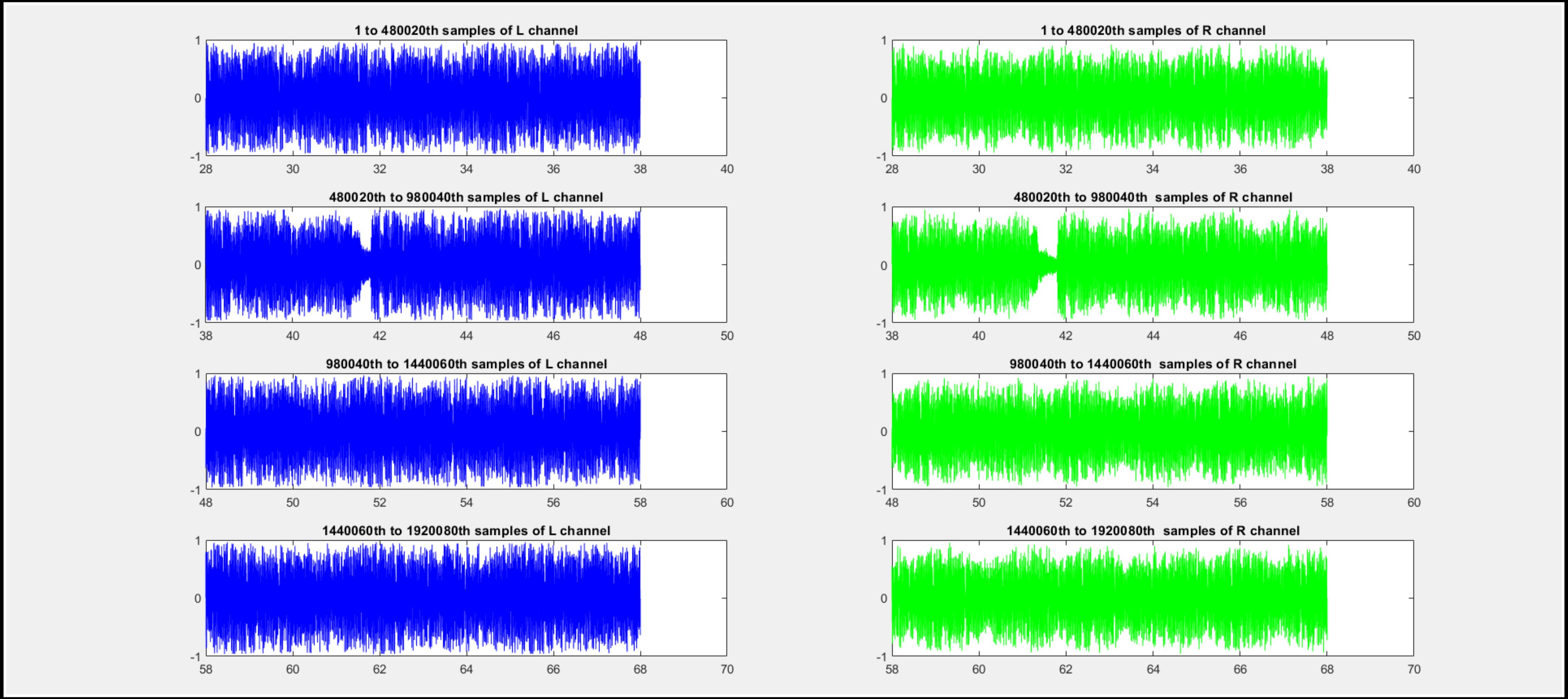
2B-1. Continued

- Once the chunks are in the matrices, it’s time to take their FFT’s
- The filter FFT is also recalculated with the length of one chunk (480020)

2B.1 - Continued

- All FFT’s have been calculated (L channel, R channel, filter),
- Now lets convolve each chunk with the filter
- After convolution, an inverse FFT is done on each resultant chunk for both channels.

```
%now that both the filter and 10 sec chunk are the same size
%with zero padding, time to convolve!

%setup again matrices to store the convolutions
overlap_fft_l = zeros(totChunks, length(chunkV_l(1,:)));
overlap_fft_r = overlap_fft_l;

% F(signal)*F(filter) for each channel
for i=1:totChunks
    overlap_fft_l(i,:) = chunkV_l_fft(i,:) .* fft_f;
    overlap_fft_r(i,:) = chunkV_r_fft(i,:) .* fft_f;
end

%now time to go back to time domain to play the filtered song!
overlap_l = zeros(totChunks, length(chunkV_l(1,:)));
overlap_r = overlap_l;

%inverse fft for each chunk and store in l and r matrices
for i=1:totChunks
    overlap_l(i,:) = ifft( overlap_fft_l(i,:) );
    overlap_r(i,:) = ifft( overlap_fft_r(i,:) );
end
```



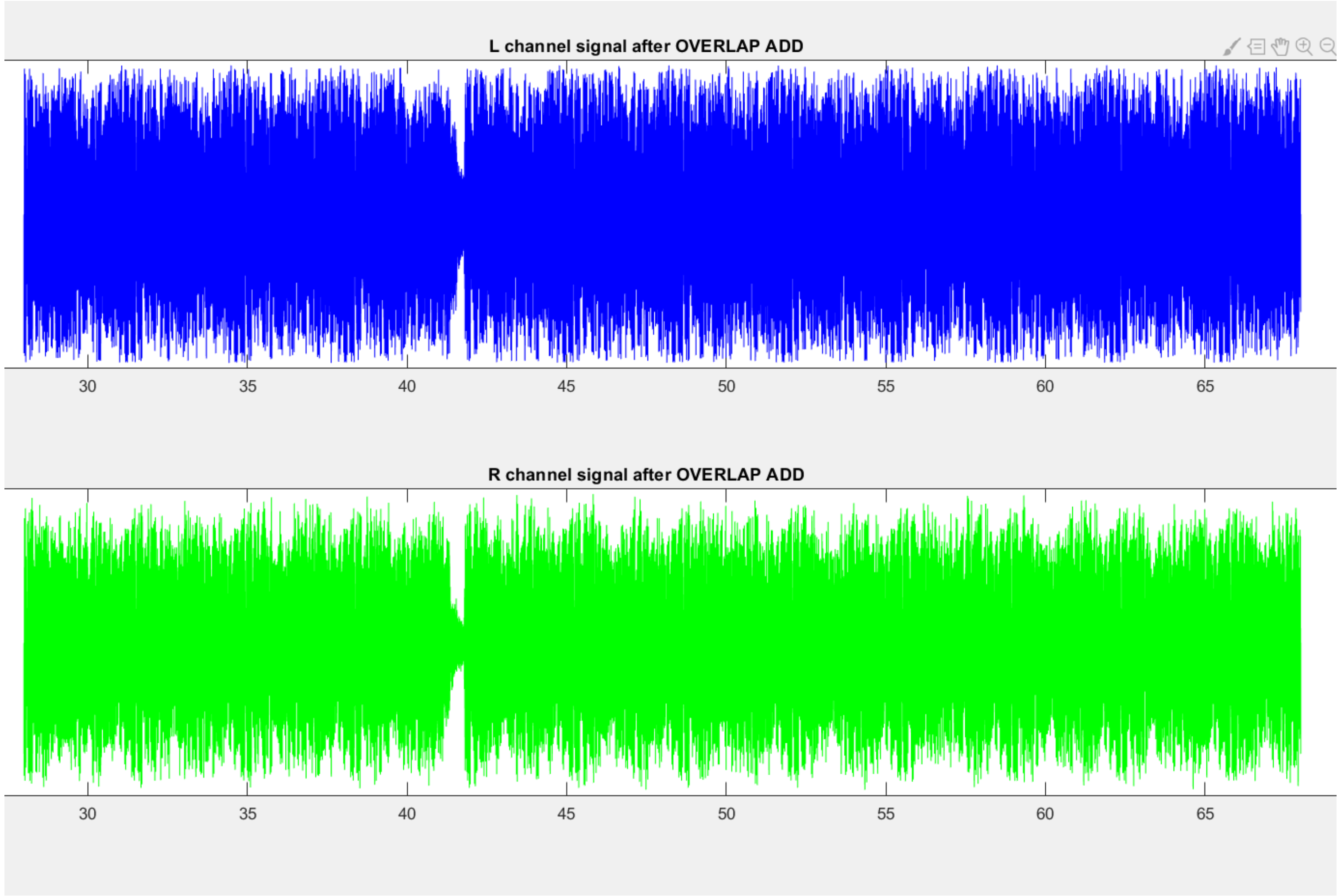
2B.1 – MATLAB Figures

2B.2 - Overlap New Signal from Chunks

- Now that all FFTS are calculated, the next step is to overlap and add.
- Here, two vectors were created to store the final convolved signal in both L and R channels.
- The algorithm adds and overlaps to the vector chunk by chunk
- Total length is M + N -1 or 1920020

fft_conv_l	1x1920020 double
fft_conv_r	1x1920020 double

```
%setup final time domain vectors for each channel
fft_conv_l = zeros(1, length(lchannel) + length(filter)-1);
fft_conv_r = fft_conv_l;

overlapLen = length(overlap_fft_l(1,:));

%now add and overlap
for i=1:totChunks-1
    if i==1
        fft_conv_l(i:overlapLen) = overlap_l(i,:);
        fft_conv_r(i:overlapLen) = overlap_r(i,:);
    end

    %here, each 10 sec chunk is being added and between 480000 and 480020,
    %overlap is being added.
    fft_conv_l(i*tenSec:i*tenSec+overlapLen-1) = ...
        fft_conv_l(i*tenSec:i*tenSec+overlapLen-1) ...
        + overlap_l(i+1,:);

    %do the same for the right channel
    fft_conv_r(i*tenSec:i*tenSec+overlapLen-1) = ...
        fft_conv_r(i*tenSec:i*tenSec+overlapLen-1) ...
        + overlap_r(i+1,:);
end
```



2B.2 – MATLAB Figures

- Convolution is correct because there are no transients in the signal.

2B.2 Confirm with RMSR Error

- Here, the variables mean the following:
- “conv_l” → FFT Convolution
- “fft_conv_l” → OVERLAP ADD METHOD

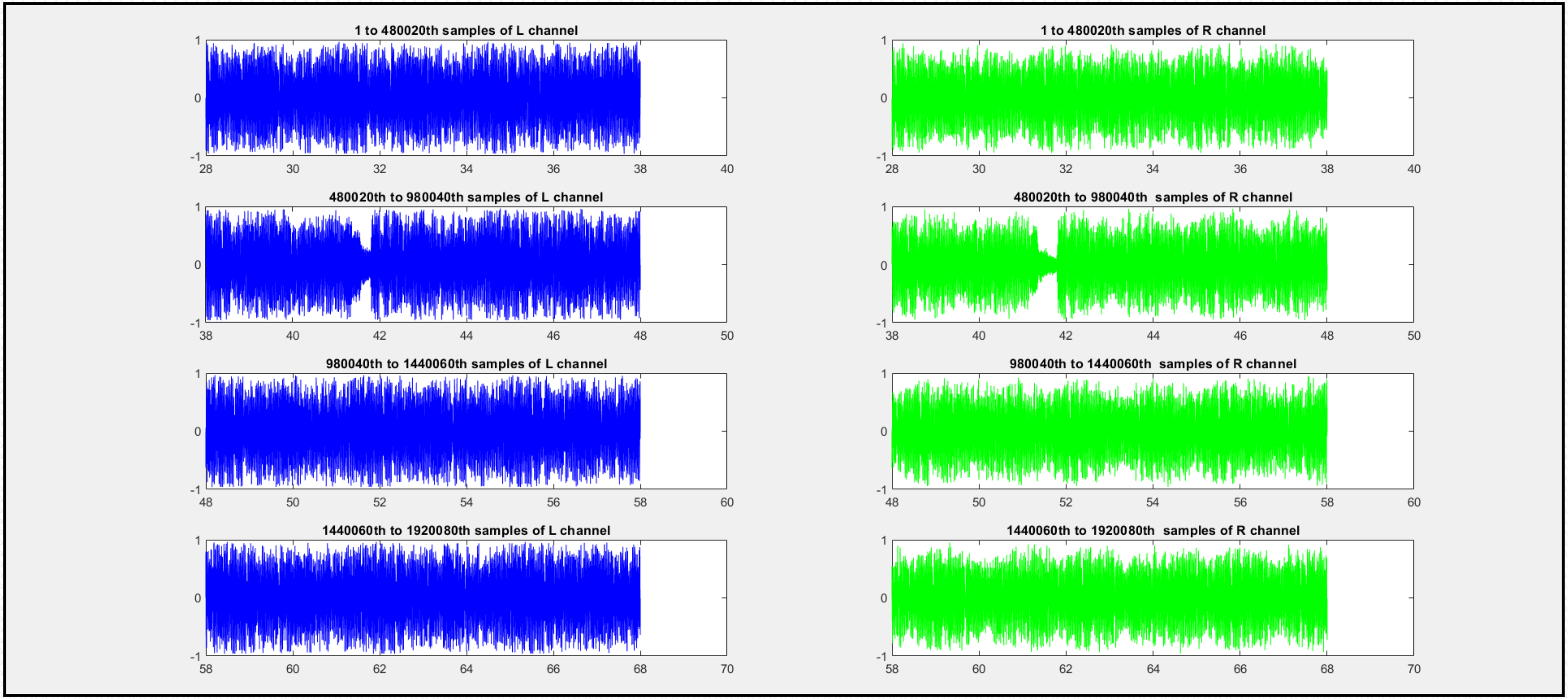
```
MSR error for Left Channel = 0.0000
MSR error for Right Channel = 0.0000
fx >>
```

- VERY LOW RMSR!!!!

```
%calculate MSR error
MSR_L = sqrt(sum((conv_l - fft_conv_l).^2 )) / length(conv_l);
MSR_R = sqrt(sum((conv_r - fft_conv_r).^2 )) / length(conv_r);

fprintf('  MSR error for Left Channel = %6.4f \n',MSR_L);
fprintf('  MSR error for Right Channel = %6.4f \n',MSR_R);
```

C – FFT Convolution

- This has been demonstrated in the previous slides so does not need further information



Conclusion – Speed of Code

```
Elapsed time is 1.465537 seconds.
Elapsed time is 0.482034 seconds.
Elapsed time is 0.717119 seconds.
Elapsed time is 0.070773 seconds.
```

1.465537 is for part A (direct calculation of convolution)

  - 2 Channels of 1.92 Million samples each = 2.62 MSPS analyzed (double precision)

- 0.482034 is for part A (FFT convolution)
  - Again 2 channels, plus same length filter (zero padded) = 5.76 Million samples = 11.949 GSPS!!!
- 0.717119 is for part B1 (FFT convolution of 8 chunks (4 per channel).
- 0.070773 is for part B2 (Custom overlap add algorithm)

References

- Filter Design

http://t-filter.engineerjs.com/
- Text on Figures

https://stackoverflow.com/questions/10525890/matlab-add-text-to-the-outside-of-figure
- DSP Guide FFT Convolution

http://www.dspguide.com/ch18/2.htm
- Class codes
- Dr. Grigoryan

- *****Can find all code on my github.com/brunogracia


THANK YOU